From Goals to Aspects: Discovering Aspects from Requirements Goal Models

Yijun Yu, Julio Cesar Sampaio do Prado Leite, John Mylopoulos
Department of Computer Science, University of Toronto

Abstract

Aspect-oriented programming (AOP) has been attracting much attention in the Software Engineering community by advocating that programs should be structured according to programmer concerns, such as "efficient use of memory". However, like other programming paradigms in their early days, AOP hasnt addressed yet earlier phases of software development. In particular, it is still an open question how one identifies aspects early on in the software development process. This paper proposes an answer to this question. Specifically, we show that aspects can be discovered during goal-oriented requirements analysis. Our proposal includes a systematic process for discovering aspects from relationships between functional and non-functional goals. We illustrate the proposed process with a case study adapted from the literature.

1. Introduction

Aspect-oriented programming (AOP) is founded on the idea of aspect [4] as a cross-cutting concern during software development. Aspects are usually "units of system decomposition that are not functional" [12], such as "no unauthorized access to data" or "efficient use of memory". Aspects cut across different components of a software system. The basic premise of aspect-oriented programming is that software structured according to aspects is easier to develop, understand and maintain.

In name and in practice, aspect-oriented programming is a programming methodology. However, this methodology does not deal with the origins of aspects. Where do aspects come from? Is there a systematic way of discovering aspects early on during the software development process? The main objective of this paper is to propose an answer to these questions. Our proposal is based on the notion of goal and the analysis techniques developed in goal-oriented requirements engineering. These techniques are shown to be useful in guiding the discovery of aspects.

Goal-oriented requirements engineering[15, 22] focuses on goals which are "roughly speaking, precursors of requirements" [7]. Goal-based models support the description and analysis of intentions that underlie a new software system. Some goal models, such as i* [25, 13], also model the actors who behold these intentions. Most variations of goal models in the literature use AND/OR trees to represent goal decomposition [14, 23] and define a space of alternative solutions to the problem of satisfying a root-level goal. There are several proposals in the literature for (formal) goal analysis techniques. For example, obstacle analysis [23] explores possible obstacles to the satisfaction of a goal. Along a different dimension, qualitative goal analysis [9] allows qualitative contributions from one goal to another, and shows how to formalize and reason with them. In whatever form, goal-oriented requirements engineering has been attracting considerable attention within the community [1, 18, 11, 2].

The rest of the paper is structured as follows. Section 2 presents a quick introduction to aspect-oriented programming, while section 3 defines a particular type of goal model, called a V-graph. Section 4 describes a systematic process for discovering candidate aspects while doing goal analysis; section 5 illustrates the aspect discovery process using a case study of media shop requirement analysis. Section 6 compares the candidate aspects with aspects found in an open-source implementation of a media shop system, and proposes abstract aspects as a way of documenting candidate aspects for later phases. Section 7 concludes the paper and sketches directions for future research.

2. Aspect Oriented Programming

An aspect, such as "efficient use of memory" is a cross-cutting concern for a software system. Dealing with code fragments that address a single aspect in different components of a software system has been a great challenge for software engineers. Structured Design [24] did recognize the importance of packing commonalities into modules. This is also known in pro-

gramming as the DRY principle, that is, "Don't Repeat Yourself". Accordingly, a good design should include components with high fan-in. Figures 1 and figure 2 show how structured design and aspect-oriented programming view the factoring out of common concerns in complementary ways.

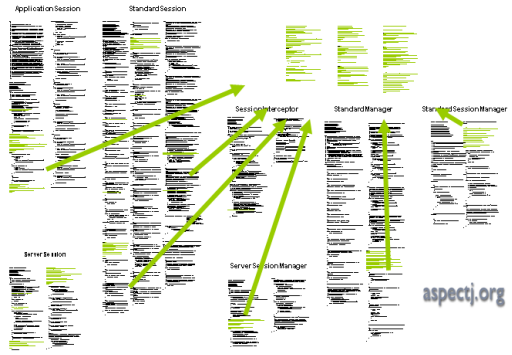

Figure 1. AOP view of aspects in source code (AspectJ).

A typical aspect expressed in AspectJ[1] syntax is as follows:

```
aspect DisplayUpdating {
  pointcut move():
       call(void FigureElement.moveBy(int, int)) ||
       call(void Line.setP1(Point))              ||
       call(void Line.setP2(Point))              ||
       call(void Point.setX(int))                ||
       call(void Point.setY(int));
  after() returning: move() {
         Display.update();
  }
}
```

The aspect *DisplayUpdating* includes the advice *Display.update()* that will be weaved in the component code after the *move()* pointcut. A pointcut is a virtual address for the inclusion of the advice in a component. This virtual address is resolved through matching. For example, every time a Line.setP1(Point) appears in a component, the advice, Display.update() will be weaved in that component.

The great benefit of software structured according to aspects is the ability to separate issues, Figure 3, and leave it to a pattern matching procedure to weave the issues together at the correct time.

Although the original AOP paper [12] pointed that aspects are mainly non-functional concerns, there has been no clear link with the notion of non-functional

[1]This example is taken from [10]

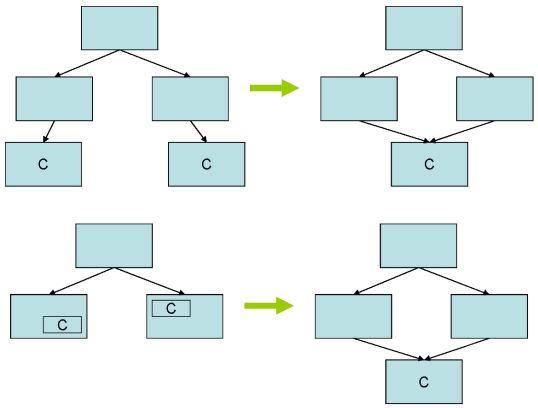

Figure 2. Don't repeat yourself (DRY) principle: increase fan-in.

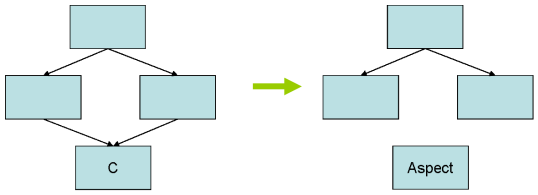

Figure 3. An aspect hides the high fan-in common part from a structured design.

requirements. [14, 5, 6] propose that non-functional requirements are first class requirements, that is, requirements that are elicited, modeled and analyzed as the software is developed. We adopt their framework and show in the rest of the paper how the analysis of functional and non-functional requirements can lead to the discovery of aspects.

3. The V-graph model

In order to reason about the interplay of functional and non-functional requirements we focus on a particular type of a goal model called a V-graph.

The V-graph is a graph with an overall shape of the letter V (Figure 4). The top two vertices of the V represent respectively functional and non-functional requirements in terms of goal models. Following [14] we represent non-functional requirements in terms of softgoals, i.e., goals with no clear-cut definition. Both models are AND/OR trees with lateral correlation links (see Table 1). The bottom vertice of the V represent tasks that contribute to the satisfaction of both goals and softgoals.

This model allows for the description of intentional nodes (goals and softgoals), as well as operational ones

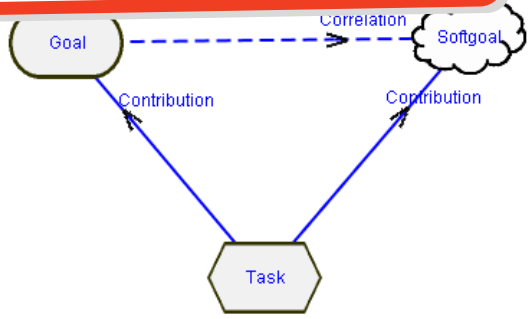

Figure 4. A V shape goal graph links a task of the goal hierarchy to a softgoal as its operationalization.

Table 1. Contribution and correlation link types.

link type	and	or	make	help	hurt	break
contrib.	Y	Y	Y	Y	Y	Y
correlat.	N	N	Y	Y	Y	Y

(tasks.) Following the NFR framework [5], we name each goal and task with two descriptors: a type and a topic. The topic is related to contextual information, such as the concepts that are important for a given application. Types have two different instantiations: for goals and tasks they describe a generic function; while for softgoals they describe a generic non-functional requirement, such as performance, safety, security, and traceability.

Figures 5 depicts two goal/softgoal trees, with correlation links among them.

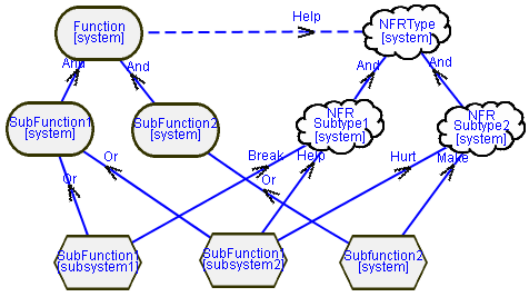

Figure 5. Decomposition of the goal and softgoals following type and topic taxonomy.

Of course, in order to produce a V-graph we need an integrated process of elicitation, modeling and analysis. The focus of this paper is on modeling and analysis, with the help of a goal analysis tool [9].

4. Building V-graphs and discovering aspects

The (manual) process we are proposing constitutes a systematic way for the refinement of a V-graph. The process ends when all root goals are satisfied and all softgoals are satisficed[2]. At this stage, we are able to identify candidate aspects by identifying tasks that have a high fan-in. The resulting graph can be further refined if candidate aspects are grouped into what we call abstract aspects.

The process of building V-graphs is iterative. During each step, the goal analysis tool is used to detect conflicts and deteriorations. A conflict occurs in a goal model when a given labeling of leaf goals as satisfied or denied leads to other goals being labeled both satisfied and denied. A deterioration occurs when the labeling of softgoals during one step of the iteration is weaker (lower) than during the previous step of the iteration.

Below we present the proposed process using a programming language-like notation.

procedure AspectFinder
input r: node /* root goal */, s: {node} /* softgoals */
output a: {aspect} and g: graph<node, link>
pre-condition $\{r\} \cup s$ are named nodes,
$a = \phi$ and $g =< \phi, \phi >$
post-condtion IsSatisfied(g, r) and IsSatisficed(g, s),
$a \neq \phi$ and $g \neq< \phi, \phi >$
begin
$g \Leftarrow$ Correlate(r, s, g)
while (**not** (IsSatisfied(g, r) **and** IsSatisficed(g, s))
or NodeToDecompose(g) $\neq \phi$)
g, *conflict* $\Leftarrow$ Decompose (g, *conflict*)
if *conflict* **then**
$g \Leftarrow$ ResolveConflict(g, *conflict*)
end while
$a \Leftarrow$ ListAspects(g)
end

AspectFinder is the root procedure. Some clarifications are need here:

1. *node* denotes a graph node, can be a goal, a task or a softgoal;

2. {*node* } denotes a set of nodes;

[2]Herbert Simon [20] used the term *satisfice* to denote the idea of "good enough" solutions to an untractable problem. The NFR framework [5] is founded on the premise that non-functional requirements (softgoals) are "satisficed" when they admit a partial, but good enough solution.

Table 2. Combinations of goal labels.

	label name	satisfice (s), denial(d)
S	satisfied	$s = 1$ and $d = 0$
D	denied	$s = 0$ and $d = 1$
U	undetermined	$s = 0$ and $d = 0$
FS	fully satisfice	$s = 1$ and $d = 0$
PS	partially satisfice	$0.5 < s < 1$ and $s + d = 1$
UN	undetermined	$s + d < 1$
PD	partially denial	$0 < s < 0.5$ and $s + d = 1$
FD	fully denial	$s = 0$ and $d = 1$
CF	conflicting	$s + d > 1$

3. { *aspect* } denotes a set of aspects;

4. and *graph<node, link>* is a graph template type in C++ terminology instantiated with *node* and *link*.

The stop condition for the iteration is defined in terms of the following predicates:

- IsSatisfied(g, r) tests if a goal r is satisfied in goal graph g, i.e., returns one of the following labels [9]: {S, D, U } and IsSatisficed(g, s) tests if softgoal s is satisficed in the goal graph g, i.e., returns one of {FS, PS, UN, PD, FD, CF } as explained in Table 2.

- NodeToDecompose(g) finds out the subset of nodes of g that are either undetermined or unsatisfied goals/tasks, or unsatisficed softgoals.

The *AspectFinder* is defined as follows.

procedure Correlate
input r: node, s: {node}, g: graph<node, link>
output g: graph<node, link>)
pre-condition $g =< \phi, \phi >$
post-condition $g =< \{r\} \cup s, L >$ where
L: {link} $\neq \phi$ and $\exists t \in s$ such that $< r, t > \in L$.
begin
for each t in s
$g \Leftarrow$ AddLink($< r, t >$, g) /*correlation link */
end
$g \Leftarrow$ MarkSatisfied(RootGoal(g))
$g \Leftarrow$ LabelPropagation(g)
end Correlate

Correlate establishes an initial relationship between root functional goals and softgoals. This relationship is represented by one or more correlation links. *Correlate* uses procedures that are defined below:

- AddLink($< n_1, n_2 >$, g) will place a link between nodes n_1, n_2 in g;

- MarkSatisfied(RootGoal(g)) will mark a root goal of g satisfied;

- LabelPropagation(g) is an algorithm described in [9] that propagates the truth labels of the leaf nodes upward to the top-level nodes according to the types of goal dependency links.

procedure Decompose
input g: graph<node, link>
output g: graph<node, link>, *conflict*: boolean
pre-condition g is consistent
post-condition g is consistent, g has more nodes
begin
$g_{\text{pre}} \Leftarrow g$
for each t: node **in** NodeToDecompose(g)
subnodes $\Leftarrow$ CreateNodes(*subnodes*)
for each s **in** *subnodes*
$g \Leftarrow$ AddNode(s, g)
if not $s \in$ NodeToDecompose(g) **then**
$g \Leftarrow$ MarkAsTask(s, g)
$g \Leftarrow$ AddLink($< s, t >$, g) /*contribution link*/
end
g, conflict $\Leftarrow$ CorrelationDecompose(g_{pre}, g, *conflict*)
end
end

Decompose refines the V-graph, using the following procedures:

- *CreateNodes* creates a set of new nodes in the graph by the human;

- *AddNode*(s, g) inserts a s into graph g;

- *NodeToDecompose(g)* gathers the goals and subgoals that have not been decomposed, or dealt with in terms of a task.

procedure ResolveConflict
input g: graph<node, link>
output g: graph<node, link>, *conflict*: boolean
pre-condition g is consistent, *conflict* = True
post-condition g is consistent, *conflict* = False
begin
$g_{\text{pre}} \Leftarrow g$
$g \Leftarrow$ RemoveConflictingLinks(g)
g, *conflict* $\Leftarrow$ CorrelationDecompose(g_{pre}, g, *conflict*)
end

ResolveConflict uses the procedure, *RemoveConflictingLinks(g)*, that removes links from a source node that has been labelled "denied" by the label propagation algorithm

procedure CorrelationDecompose
input g_{pre}, g: graph<node, link>
output g: graph<node, link>, *conflict*: boolean
pre-condition g_{pre} is consistent
post-condition g is consistent
begin
 $g \Leftarrow$ MarkGoalGraph(g,{U, S, D })
 while (CorrelationLinkToDecompose(g_{pre}) $\neq \phi$
  or IsNotConsistent(g_{pre}, g))
  $l \Leftarrow$ GetACorrelationLink(g_{pre}, l)
  $g \Leftarrow$ RemoveLink(l, g)
  $g \Leftarrow$ AddLinks(CreateLinks(l), g)
  $g \Leftarrow$ LabelPropagation(g)
 end
 g, *conflict* $\Leftarrow$ LabelPropagation(g)
end

CorrelationDecomposition uses the procedures that are, briefly, described below:

- *MarkGoalGraph*(g, ...) initializes task nodes (associated to leaf goal nodes) with a satisfied/denied label specified by the user, and all other goal nodes as undetermined to facilitate the label propagation algorithm.
- *CreateLinks*(l) where $l =< r, s >$ creates new links between subgoals of r and subsoftgoals of s specified by the user;
- *CorrelationLinkToDecompose* looks for all the correlation links except for those from tasks to leaf softgoals.

procedure Deteriorates
input g_{pre}, g: graph<node, link>
output *noDeterioration*: boolean
begin
 $r_0, s_0 \Leftarrow$ RootGoal(g_{pre}), Softgoals(g_{pre})
 $r, s \Leftarrow$ RootGoal(g), Softgoals(g)
 noDeterioration $\Leftarrow$ IsSatisfied(g_{pre}, r_0) **and** IsSatisfied(g, r)
  and $\forall n \in s_0 \cap s$:
 LessThan(IsSatisficed(g_{pre}, n), IsSatisficed(g, n))
end

Within the *Deteriorates* procedure, *LessThan*(label$_1$, label$_2$) compares two labels according to the label order (`FD < PD < UN < PS < FS`).

procedure ListAspects(g)
input g: graph<node, link>
output *as*: {aspect}
pre-condition In the goal graph g, all goals are satisfied and all softgoals are satisficed. There is no conflict and all correlation links are consistently decomposed into contribution links from tasks to softgoals.
post-condition $as \neq \phi$
begin
 $as \Leftarrow \phi$
 for each n **in** Softgoals(g) such that
  n is the direct parent of a task
  a: aspect $\Leftarrow$ CreateAspect(n, a)
  for each $< t, n >\in$ Links(g) where $t \in$ Tasks(g)
   $a \Leftarrow$ AddAdvice(t, a)
   for each $f \in$ CrosscuttingGoals(t, g)
    $a \Leftarrow$ AddToPointcuts(f, a)
   end
  end
  $as \Leftarrow as \cup\{a\}$
 end
end

ListAspects(g) gathers a set of tasks that contribute to a softgoal, that is, there is a contribution link $< t, s >$ and more than one chain of contribution links $\{t \rightarrow, \cdots, \rightarrow f\}$ where t is a task, s is a softgoal, f is the functional goal crosscutted by t.

Figure 6 shows a pictorial view of the process of modeling V-graphs. Initially, a set of objectives including one or more functional goals and several non-functional softgoals are listed. Then requirements engineers and domain experts decompose goals (softgoals) into subgoals (subsoftgoals) or tasks, and correlate the goals/tasks from the functional perspective to the softgoals/operationalizations from the non-functional one. The refinement process must be monotonic (no deteriorations) and resolve conflicts through a formal goal analysis until goals are satisfied and softgoals are satisficed.

5. Analyzing V-graphs to discover aspects: An Example

We use the Media Shop example to illustrate the application of our procedure. The Media Shop example has been used in the context of intentional modeling [3] and is a good example to show the interplay of functional and non-functional goals. Through Figures 7

Figure 6. Discover aspects during the goal oriented requirement engineering

to 12 we show several steps of the *AspectFinder* process. Figure 13 presents the final graph and Figure 14 presents the output of the analysis tool for the final graph. Finally, the candidate aspects can be seen in the center-right part of the final graph, Figure 13, that is they are the operationalizations of the softgoals and the relations to functional goals.

Once we have the elicited information regarding the goals we can start the process. We start by listing root goals and softgoals. Later we apply the *Correlate* procedure (see Section 3) to add correlation links. Figure 7 shows this for the Media Shop example.

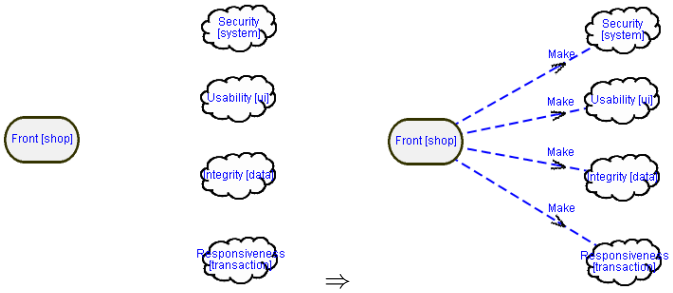

Figure 7. Correlate

Given this graph, we invoke the *LabelPropagation* procedure, which return FS (s=1, d=0) values for all the nodes. This means that the graph is not deteriorating and has no conflicts, as shown in the right part of Figure 8. Here we export the goal analysis result to the `graphviz` tool [8] for the layout.

However, if we decompose the goal above using *Decompose*, which propagates correlations through *CorrelationDecompose*, as shown in the left part of Figure 9,

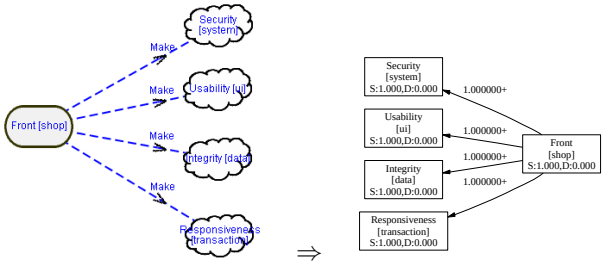

Figure 8. Consistent Graph

this results in a deterioration, since the softgoal *Responsiveness[transaction]* becomes partially satisficed (PS) after decomposition and was previously, in Figure 8, fully satisficed (FS).

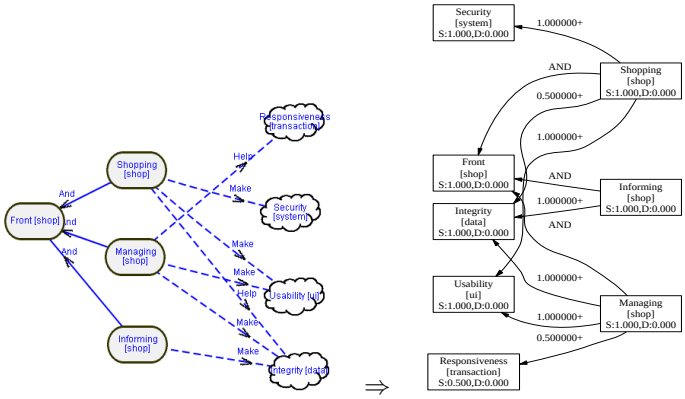

Figure 9. Detection of Inconsistent Decomposition

With the feedback provided by the goal analysis tool, (see Figure 6), the requirements engineer proposes a different decomposition Figure 10) which removes the deterioration.

At the left part of Figure 11 we show a part of the Media Shop graph where the transaction goals are being decomposed into two tasks that by the correlation links are related by "make" and "hurt" to the same softgoal. The right part of the figure will show the result of the *LabelPropagation* procedure. Here we can see a conflicting label for the softgoal "Responsiveness [Transaction]": S=1 and D=1.

With this feedback, the requirements engineer is able to change the graph into a well formed graph, by removing the links that caused the conflict.

As indicated earlier, Figure 13 is the V-graph for the Media Shop example [3] We can now apply the procedure ListAspect presented in Section 3. Table 3 shows

[3]For space reasons, we only show part of the example.

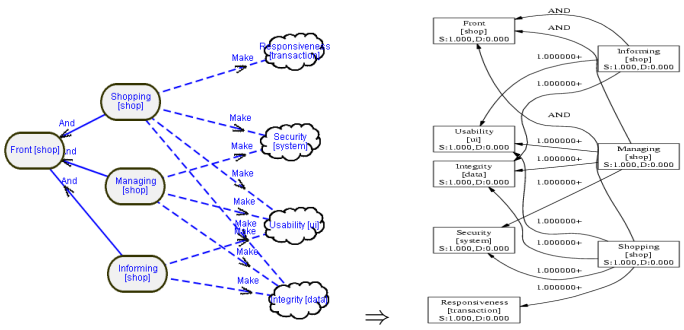

Figure 10. Decomposition with no deterioration

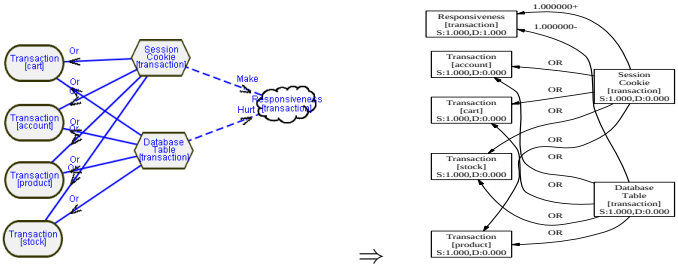

Figure 11. Conflict detection for the softgoal "Responsiveness [transaction]"

the aspects (the operationalized softgoals, related functional goals, as well as related functional tasks). These aspects are named after the operationalized softgoal.

6. Discussion

As shown from Table 3, the proposed process was able to identify candidate aspects. However, how can we be sure that these candidate aspects that we have listed are reasonable?

Apart from appealing to common sense, we have also attempted an evaluation by matching these aspects against an open source implementation of the Media Shop example. The source code is available at *OSCommerce* [16], and its core contains about 65 KLOC in PHP.

After a thorough clone detection using Semantic Designs' clone detector (`CloneDR`), it found that the high fan-in aspects are among the 463 clone tuples found in the weaved source code. Out of the 52036 LOC under the clone detection analysis, 16833 LOC or 19.1% of the total code are found as scattered clones [19].

It is interesting to note that the goal graphs obtained from the proposed process can be simplified by factoring out candidate aspects, as per Figure 15.

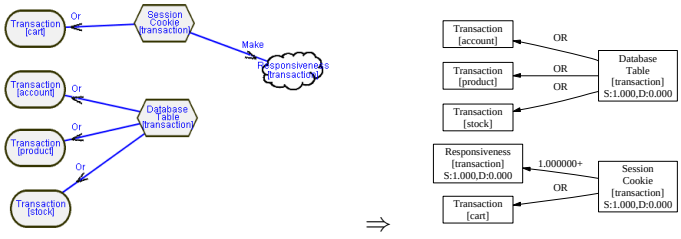

Figure 12. Resolving conflicts

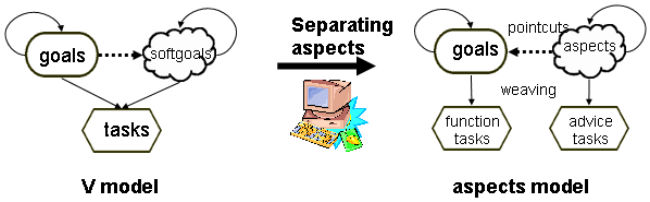

Figure 15. Separating aspects advised tasks from the functional tasks.

In that sense, we are proposing *abstract aspects* as an abstraction of aspects, intended to help requirements engineers with respect to the scalability of their models. Abstract aspects may defined in terms of a syntax such as that shown below.

```
abstract aspect Responsiveness[transaction] {
   pointcut frequentTransaction():
      and(Preparing[cart,product]) ||
      and(CheckingOut[cart, product, account, stock]);
   required () by: frequentTransaction() {
      SessionCookie[transaction]();
   }
}
```

Here the abstract aspect *Responsiveness[transaction]* has the advice task *SessionCookie[transaction]* that is required by the *frequentTransaction()* pointcut. In the description of this pointcut, we will look for addresses that match the two matching conditions `and(Preparing[cart,product])`, `and(CheckingOut[cart, product, account, stock])` . Note that Figure 13 is already a weaved graph. As such, the description above can be traced. However, Figure 16 shows the goal graph as if the above abstract aspect was applied. The notion of abstract aspect clearly needs further research.

7. Conclusions

We have proposed a systematic process whereby aspects can be discovered by conducting goal analysis for a system-to-be. We have demonstrated the process

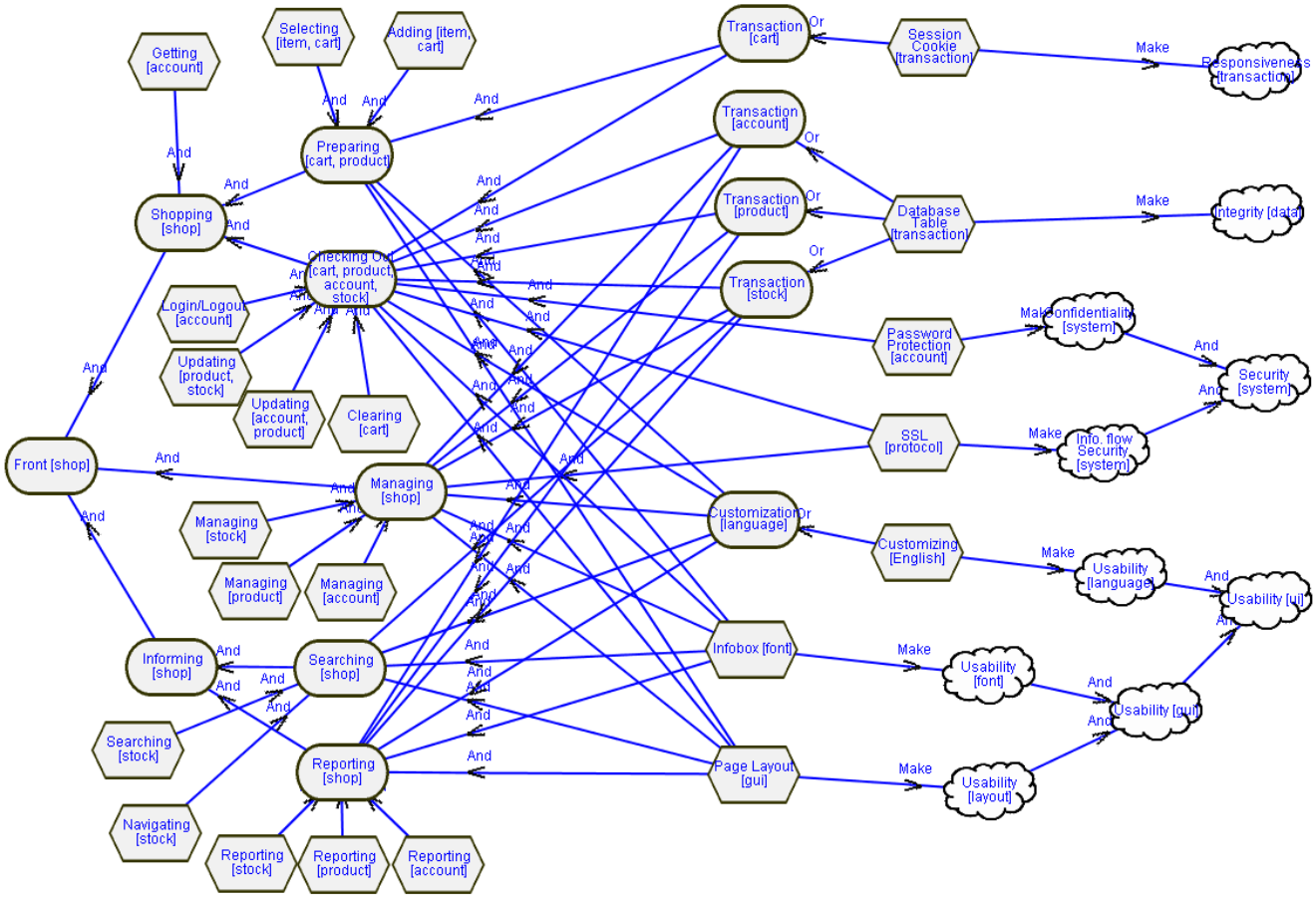

Figure 13. The final media shop goal graph after removing conflicts.

Table 3. Candidate aspects discovered for Media Shop example.

Aspect softgoal	Advising task	Crossing-cutting functional goals
Responsiveness [transaction]	SessionCookie [transaction]	Preparing [cart, product], Checking Out [cart, product, account, stock]
Integrity [data]	DatabaseTable [transaction]	Checking Out [cart, product, account, stock], Managing [shop], Searching [shop], Reporting [shop]
Confidentiality [system]	Password Protection [account]	Checking Out [cart, product, account, stock]
Info. Flow Security [system]	SSL [protocol]	Checking Out [cart, product, account, stock], Managing[shop]
Usability [language]	Customizing [English]	Preparing [cart, product], Checking Out [cart, product, account, stock], Managing [shop], Searching [shop], Reporting [shop]
Usability [font]	Infobox [font]	Preparing [cart, product], Checking Out [cart, product, account, stock], Managing [shop], Searching [shop], Reporting [shop]
Usability [layout]	Page Layout [gui]	Preparing [cart, product], Checking Out [cart, product, account, stock], Managing [shop], Searching [shop], Reporting [shop]

DRAFT

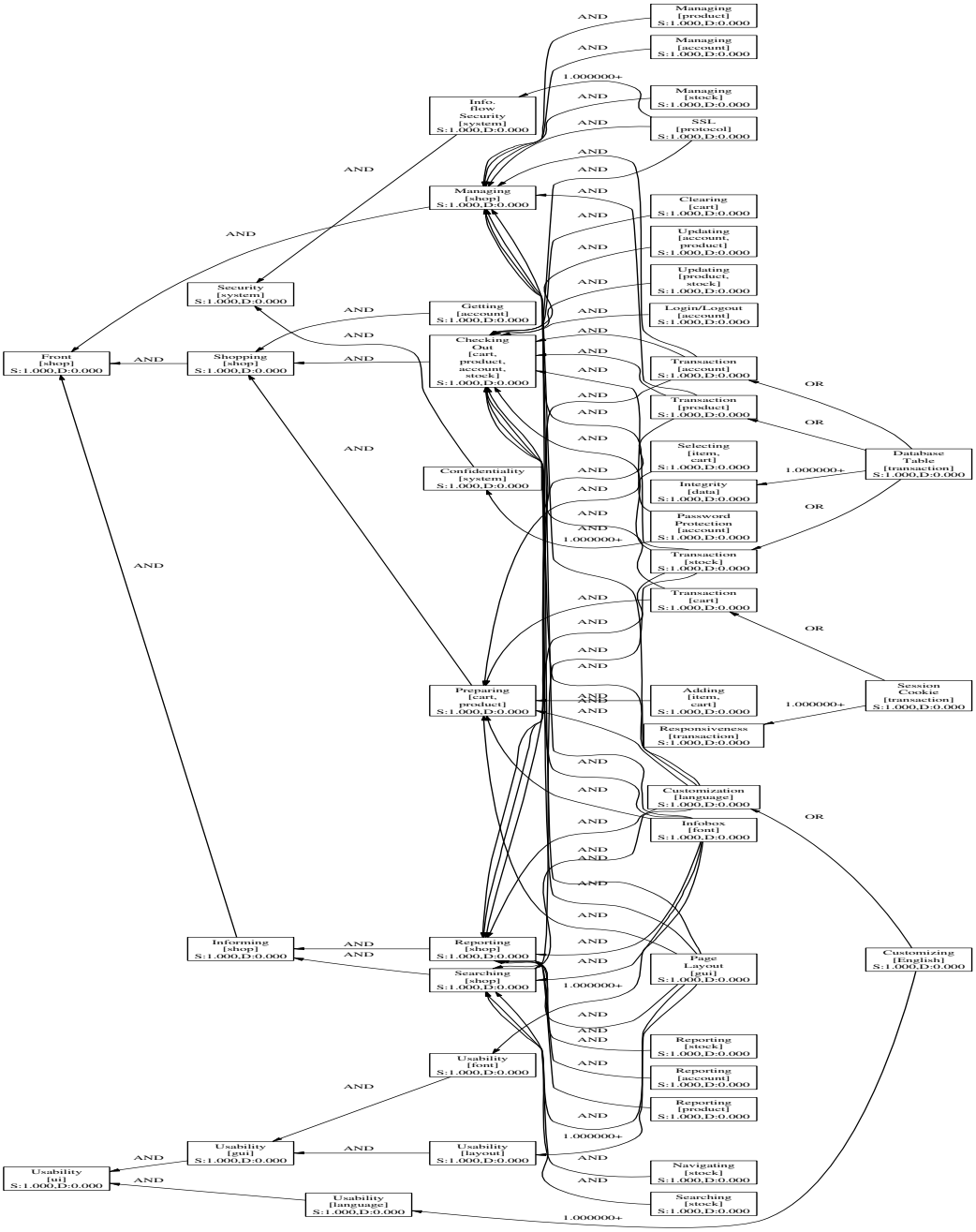

Figure 14. The evaluation result of the goal graph in Figure 13. All goals/tasks are satisfied and all softgoals and operationalizations are satisficed and there are no conflicts.

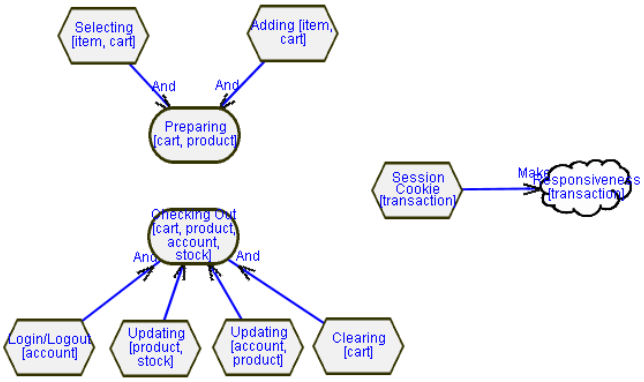

Figure 16. Separating the advicing task of the abstract aspect *Responsiveness[transaction]* **from the functional goals in its pointcut.**

with a case study adopted from the literature, and we have compared the resulting aspects with those found in an independently-developed open source solution.

Compared to [21], we are providing a well-defined, tool-supported process that they assume must be done by an experienced software engineer. Compared to [17], we are treating aspects at a higher level of abstraction, in terms of goals, and doing so in a systematic way based on formal analysis.

For future research, we plan to investigate further the notion of abstract aspects, especially so in the context of software reusability, also in reengineering legacy code to make it aspect-oriented.

Acknowledgement

We thank Dr. Ira Baxter (Semantics Designs) for the results reported on section 6 regarding the analysis of osCommerce source code.

References

[1] A. I. Anton, R. A. Carter, A. Dagnino, J. H. Dempster, and D. F. Siege. Deriving goals from a use-case based requirements specification. *Requirement Engineering*, 6(1):63–73, 2001.

[2] D. Bolchini, P. Paolini, and G. Randazzo. Adding hypermedia requirements to goal-driven analysis. In *RE 2003*, pages 127–137, 2003.

[3] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information Systems*, 27(6):365–389, 2002.

[4] v. F. C. Chavez, F. A. Garcia, and C. J. P. Lucena. Tutorial: Desenvolvimento de Software Orientado a Aspectos" (in Portuguese). In *The 17th Brazilian Symposium on Software Engineering SBES, (http://www.sbbd.fua.br/inenglish/paginaseng/sbes-tutorials.htm)*, 2003.

[5] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishing, 1999.

[6] L. M. Cysneiros, J. C. S. do Prado Leite, and J. de Melo Sabat Neto. A framework for integrating non-functional requirements into conceptual models. requirements engineering. *Requirement Engineering*, 6(2):97–115, 2001.

[7] M. S. Feather, T. Menzies, and J. R. Connelly. Relating practitioner needs to research activities. In *RE 2003*, pages 352–, 2003.

[8] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE TRANS. SOFTW. ENG.*, 19(3):214–230, May 1993.

[9] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. *LNCS*, 2503:167–??, 2002.

[10] E. Hilsdale and G. Kiczales. Aspect oriented programming with AspectJ, xerox parc, http://www.aspectj.org.

[11] H. Kaiya, H. Horai, and M. Saeki. Agora: Attributed goal-oriented requirements analysis method. In *RE 2002*, pages 13–22, 2002.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect oriented programming. *LNCS*, 1241:220–242, Oct. 1997.

[13] L. Liu, E. Yu, and J. Mylopoulos. Security and privacy requirements analysis within a social setting. In *RE 2003*, pages 151–161, 2003.

[14] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, June 1992.

[15] J. Mylopoulos, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, Jan. 1999.

[16] pair Networks. oscommerce: Open Source E-Commerce Solutions, http://www.oscommerce.com.

[17] A. Rashid, P. Sawyer, A. M. D. Moreira, and J. Arajo. Early aspects: A model for aspect-oriented requirements engineerin. In *RE 2002*, pages 199–202, 2002.

[18] C. Rolland and N. Prakash. From conceptual modelling to requirements engineering. *Annals of Software Engineering*, 10:151–176, 2000.

[19] Semantics Designs. CloneDR, http://www.semdesigns.com/products/DMS.

[20] H. A. Simon. *The Science of the Artificial, 3rd Edition*. MIT Press, 1996.

[21] G. Sousa, I., and J. Castro. Adapting the NFR framework to aspect-oriented requirement engineering. In *The XVII Brazilian Symposium on Software Engineering, Manaus, Brazil, October, 2003*, 2003.

[22] A. van Lamsweerde. Goal-oriented requirements engineering: From system objectives to UML models to precise software specifications. In *ICSE 2003*, pages 744–745, 2003.

[23] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Software Eng.*, 26(10):978–1005, 2000.

[24] E. Yourdon and L. L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 1st ed.* Prentice-Hall, 1979.

[25] E. S. K. Yu and J. Mylopoulos. From E-R to A-R – modelling strategic actor relationships for business process reengineering. *Int. Journal of Intelligent and Cooperative Information Systems*, 4(2–3):125–144, 1995.